Secured Information Flow for Asynchronous Sequential Processes

Isabelle Attali, Denis Caromel, Ludovic Henrio, Felipe Luna

Secured Information Flow for Asynchronous Sequential Processes

Isabelle Attali, Denis Caromel, Ludovic Henrio[1],
Felipe Luna Del Aguila

INRIA Sophia Antipolis, CNRS - I3S - Univ. Nice Sophia Antipolis
2004, Route des Lucioles, BP 93 - F-06902 Sophia Antipolis Cedex, France
{ia, caromel, henrio, fluna}@sophia.inria.fr

Abstract

We present in this article a precise security model for data confidentiality in the framework of ASP (Asynchronous Sequential Processes). ASP is based on active objects, asynchronous communications, and data-flow synchronizations. We extend it with security levels attached to activities (active objects) and transmitted data.

We design a security model that guarantees data confidentiality within an application; this security model takes advantages of both mandatory and discretionary access models. We extend the semantics of ASP with predicate conditions that provide a formal security framework, dynamically checking for unauthorized information flows. As a final result, all authorized communication paths are secure: no disclosure of information can happen. This theoretically-founded contribution may have a strong impact on distributed object-based applications, that are more and more present and confidentiality-demanding on the Internet, it also arises a new issue in data confidentiality: authorization of secured information flow transiting (by the mean of futures) through an unsecured component.

Key words: Access control, distribution, objects, futures.

1 Introduction

The main contribution of this work is to provide data confi dentiality and secure information fbws for asynchronous distributed object-based applications. The proposed security model heavily relies on security policy rules with mandatory enforcements for the control of information fbw. While information fbws are generally verifi ed statically [20,3,15,14,25,22,16,10], our attention is focused on dynamic verifi cations. To achieve it, our model has an information control policy that

[1] University of Westminster - Harrow School of Computer Science - Harrow, HA1 3TP (UK)

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

includes discretionary rules, and because these rules are by nature dynamically enforceable, we can take advantage of the dynamic checks to carry out at the same time all mandatory checks. As another advantage of this approach, dynamic checks do not require to modify compilers, do not alter the programming languages, do not require modifications to existing source codes, and provide flexibility at run-time. Thus, dynamic checks fit well in a middleware layer which, in a non-intrusive manner, provides and ensures security services to upper-level applications.

Our underlying programming model [8] is based on active objects, asynchronous communications, and data-flow synchronizations. On the security side, security levels are used to independently tag the entities involved in the communication events: active objects and transmitted data. These "independent" tagging is however subject to discretionary rules. The combination of mandatory and discretionary rules allows to relax the strict controls imposed by the mandatory rules.

Overview

As activities do not share memory, our security mechanism associates a security level to each activity (an activity is a set of objects manipulated by a single thread). Additionally, a level can be associated to data transmitted between activities (request parameters and created activities). Our mechanism is based on the strict notions of "no write-down" and "no read-up" taken from the model of Bell-LaPadula [4], but discretionary rules allow to associate specific level to communicated data; thus avoiding restrictiveness of the above notions. Dynamic checks are performed at activity creation, request and reply communications. The advantages of this approach are twofold:

a sound foundation. This security model is founded on a strong theoretical background, the ASP calculus [7,6], related to well-known formalisms [14,13,10,9]. We extend the formal semantics of ASP with predicate conditions. This provides a formal basis to our model, and makes it possible to dynamically check for unauthorized accesses. Finally, in order to prove the correctness of our security model, an intuitive secure information flow property is defined and proved to be ensured by the application of access control model.

scalability and flexibility. We also target practical use of this model, with an implementation into middlewares, e.g. ProActive [21]. The granularity of our security model is defined in order to make it both efficient (because there is no security check inside an activity) and finely tunable: levels can be defined on activities but a specific level can be given to communicated data.

The impact of this work lies on the recent changes of paradigms in the area of distributed computing. The service oriented nature of ASP makes communications asymmetric (request and replies) and asynchronous. This security framework is, to our knowledge, the first to be adapted to the specificities of these communications. One of the main novel issues revealed by this work is the case of the first class futures (futures are promises of reply), and the fact that some *secured* information flows, impossible in other security frameworks, can be envisioned for secured

asynchronous services.

Section 2 recalls our base model for objects and communications. Section 3.1 presents an access control model that can has been implemented for ASP. Section 3.2 defines and verifies an intuitive secure information flow property: an information flow is secure if all activity creations, requests and replies transmissions are secure. Section 3.3 analyzes the specificity of our security model for service-oriented computing. Next, relation to existing work is discussed in section 4. An appendix gives an example showing the kind of security policy we are able to ensure, and the specificity of our model.

2 A Model for Objects and Communications

distributed over (single threaded) activities, without shared memory. Activities communicate by asynchronous method calls, and reply by means of futures, thus leading to a data-flow synchronization. Futures allow latency hiding without loosing, neither ease of programming nor efficiency. This paper will soundly formalize the security of information flow for this recent programming methodology: asynchronous service-oriented programming.

2.1 Distributed Object Model

The ASP calculus is an extension of the $\mathbf{imp}\varsigma$-calculus [1] where asynchronous communicating processes prevail. These processes, or *activities*, are running in parallel, but with their internal operations executed sequentially. What makes it outstanding are the concepts of *active objects*, *wait-by-necessity*, and *futures*. An *active object* is a "classical" object which is to be run remotely, in an activity, with its own sequential thread of execution. Table 1 presents the syntax of the ASP language. The classical sequential reduction rules for the semantics description can be found in [7].

$a, b \in L$'	$::= x$	variable,
	$\mid [l_i = b_i; m_j = \varsigma(x_j, y_j)a_j]_{j\in 1..m}^{i\in 1..n}$	object,
	$\mid a.l_i$	field access,
	$\mid a.l_i := b$	field update,
	$\mid a.m_j(b)$	method call,
	$\mid clone(a)$	superficial copy,
	$\mid Active(a, m_j)$	object activation,
	$\mid Serve(m_1, .., m_n)^{n>0}$	service primitive,
	$\mid \iota$	location (not in source).
	$\mid a \Uparrow f, b$	a with continuation b

Table 1
The ASP calculus syntax

It is important to note that an activity is composed of only one active object, many passive (i.e., classical) objects and also many references to other active ob-

jects. As an example, Figure 1 shows activities α and β, where activity α has (among other entities) its own active object, two passive objects and a reference to activity β. Additionally, there is no shared memory: passive objects can only be referenced by objects belonging to the same activity, but any object can reference an active one.

One of the main contribution of ASP is the formalization of *futures* and request-reply patterns of communication. Futures are generalized references representing promises of reply that can be manipulated as a classical object (i.e. copied and transmitted inside and between activities) while their real value is not needed. An operation that needs the value of the object (e.g. a field access) is blocked until the necessary reply occurs. This automatic and transparent synchronization mechanism is called *wait-by-necessity*.

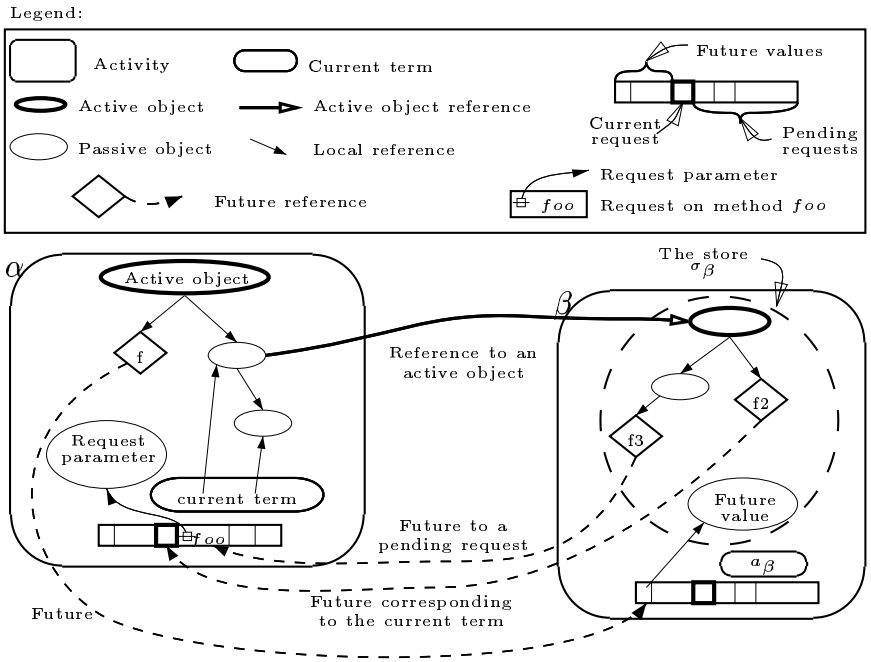

Fig. 1. Example of a parallel configuration

Recalling the ASP syntax and semantics, a parallel configuration is a set of activities (α, β, $\gamma \in$ *Act*) of the form: P, Q ::= $\alpha[a; \sigma; \iota; F; R; f] \parallel \beta[\cdots] \parallel \cdots$ where *a* is the current term to be reduced; σ is the store containing all objects belonging to activity α; ι is the active object location; *F* is the list associating each result of a served request to its *futures*; *R* is the list of pending requests; and *f* is the future of the current term. Additional semantic notations are: locations ι and future identifiers f_i are local to an activity; a future $f_i^{\alpha \to \beta}$ is defined by an identifier f_i, a source activity α and a destination activity β according to the request; a reference to the active object of activity α is denoted by $AO(\alpha)$; and a reference to a future is denoted by $fut(f_i^{\alpha \to \beta})$.

Informally, ASP semantics consists in partitioning objects into distinct activi-

ties manipulated by a single thread (no memory is shared). Those activities communicate by asynchronous request-response mechanism: When an object perform a method call an an active object (i.e., remote object), a request is enqueued on the callee side and the caller receives a future (an empty object that will be fi lled with the result of the request).

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma') \qquad \rightarrow_S \text{does not clone a future}}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P} \ (\textsc{local})$$

$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \\ \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \qquad \sigma_\gamma = copy(\iota'',\sigma)\end{array}}{\begin{array}{c}\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[\iota''.m_j();\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P\end{array}} \ (\textsc{newact})$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \\ \iota_f \notin dom(\sigma_\alpha) \qquad \sigma'_\beta = Copy\&Merge(\sigma_\alpha,\iota'\ ;\ \sigma_\beta,\iota'') \\ \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}} \ (\textsc{request})$$

$$\frac{R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M,\ m \notin R'}{\begin{array}{c}\alpha[\mathcal{R}[Serve(M)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \\ \alpha[\iota.m_j(\iota_r) \Uparrow f,\mathcal{R}[[]];\sigma;\iota;F;R' :: R'';f'] \parallel P\end{array}} \ (\textsc{serve})$$

$$\frac{\begin{array}{c}\iota' \notin dom(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \\ \sigma' = Copy\&Merge(\sigma,\iota\ ;\ \sigma,\iota')\end{array}}{\alpha[\iota \Uparrow f',a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P} \ (\textsc{endservice})$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = fut(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \\ \sigma'_\alpha = Copy\&Merge(\sigma_\beta,\iota_f\ ;\ \sigma_\alpha,\iota)\end{array}}{\begin{array}{c}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}} \ (\textsc{reply})$$

Table 2
Parallel reduction

2.2 Communication Model

The ASP communication model is based on the parallel reduction rules shown in Table 2. These rules are based on a $copy(\iota,\sigma)$ operator which performs a deep copy of the store σ starting at location ι and $Copy\&Merge(\sigma_\beta,\iota'\ ;\ \sigma_\alpha,\iota)$ which appends, at the location ι of the store σ_α, a deep copy of the store σ_β starting at location ι'.

From these reduction rules, only the communication rules (NEWACT, REQUEST and REPLY) are involved in the security framework. The NEWACT reduction rule

creates a new activity γ containing the deep copy of an object. A generalized reference to this activity $AO(\gamma)$ is stored in the source activity α. In the REQUEST reduction rule, activity α sends a new request to activity β. The new request $[m_j; \iota''; f_i^{\alpha\to\beta}]$ is made up of the target method m_j, the location ι'' of the argument passed in the request message, and the future identifier $f_i^{\alpha\to\beta}$ which will be related to the response resulting from the request. Note that in location ι'' there is a deep copy of the argument passed to the target method. The REPLY reduction rule, takes a reference to a future and updates it with its value. The reference to the future must exist in one activity α and the corresponding value must have been calculated in another activity β. Note that a future $f_i^{\gamma\to\beta}$ can be updated in an activity different from the origin of the request ($\gamma \neq \alpha$).

3 The Security Model

In this section, we define a security model that guarantees the classic property of data confidentiality for multi-level security systems: a specific user with the appropriate clearance will be given access only to the information that he/she is allowed to handle. This notion of data confidentiality must not be confused with the confidentiality provided by encryption mechanisms (i.e. information obscuring). The security terminology uses the subjects-objects relationship [24], and because the word "object" can be confusing here, we refer to this relationship as "subject-target".

We first recall usual notions of access control models and define our model in terms of entities and secured communications; which means that we formally extend ASP into Secure ASP. Then, we present our notion of secure information flow between activities with an important property for data confidentiality. We finally point out a fundamental aspect of the expressiveness of our model for service-oriented computing.

3.1 Access Control Model

Access control models are generally classified into mandatory (MAC) or discretionary (DAC) models. The MAC model can be best described through the Multi-Level Security (MLS) model which is based on a lattice of security levels assigned to subjects and targets. Once levels are assigned, neither "normal users" nor processes can change them; making the system more secure against unauthorized access to the information. The MLS model is suited to address the confidentiality issue in information flows, but its inconvenience is that in certain cases it is less than adequate for practical systems. DAC models are based on an access control matrix relating rights on subjects over targets, where the rights may be assigned at the "discretion" of the "normal users" or their processes; this simple form of operation makes it more flexible compared to MLS.

The solution we propose is based on the concepts of MLS, with analogous notions of "no write-down" and "no read-up" taken from the model of Bell-LaPadula [4],

but it is generalized here by introducing exceptions according to discretionary rules.

We begin by describing all entities involved in our security framework:

- $\mathcal{S}$ is the set of activities acting as subjects and/or targets: $\alpha, \beta, \gamma, ... \in \mathcal{S}$;
- $\mathcal{D}$ is the set of objects sent in the arguments of REQUESTS; a REQUEST is now written as $Rq_{\alpha\to\beta}(d)$ where $d = \sigma_\alpha(\iota') \in \mathcal{D}$,
- $\mathcal{R}$ is the set of objects associated to futures, and returned in REPLIES; a REPLY is now written as $Rp_{\beta\to\alpha}(r)$ where $r = \sigma_\beta(\iota_f) \in \mathcal{R}$.

Let $\mathcal{A}$ be the set of actions involved in the security mechanisms, i.e., REQUEST, REPLY and NEWACT. The following notations are added to ASP:

- Security levels λ are taken from a finite set $\mathcal{L}$, partially ordered by the relation $\leq$, $\forall i \in \mathcal{S} \cup \mathcal{D} \cup \mathcal{R}$, $\lambda_i \in \mathcal{L}$,
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{A}$ represents the authorized actions (source, destination, action),
- the matrix $\mathcal{M} : \mathcal{S} \times \mathcal{S} \to \mathcal{P}(\mathcal{A})$ gives explicit (discretionary) rights to assign a level to a given data for a given action. For each subject-target pair, the matrix contains a set of authorized actions involving the assignation of a security level. $\mathcal{P}(\mathcal{A})$ classically denotes the set of sets of actions. The classic subject-target-action matrix is extended to include (allowed) security levels. This matrix can be implemented with a security policy file (e.g. XACML policy files [11]).

This results in the following characterization of actions:

- $Nw(\gamma, \lambda_\gamma)$ is a modified activity creation rule (NEWACT) in order to assign a security level λ_γ to a created activity γ,
- $Rq_{\alpha\to\beta}(d, \lambda_{in})$ is a modified REQUEST transmission rule tagging the transmitted data with a security level (the programmer gives a security level to data d),
- $Rp_{\beta\to\alpha}(r)$ is a REPLY transmission rule unchanged from original ASP.

To summarize, $a \in \mathcal{A}$ if and only if $a = Rq_{\alpha\to\beta}(d, \lambda_{in}) \vee a = Rp_{\beta\to\alpha}(r) \vee a = Nw(\gamma, \lambda_\gamma)$. Moreover, to be precise, $\mathcal{M}$ only has values in REQUEST or NEWACT. That is to say, it is not possible in our model to give to replies a discretionary right.

The security levels of subjects, targets, data and responses reflect the form of communications handled by activities. Figure 2 shows this form of communications. All activities are tagged with a security level and all objects and their methods therein contained will automatically inherit that level (a and b are objects, m_s is the calling method of the source activity, and m_t is the target method). Every data d used in a request transfer is also marked with a security level but this level is independent from that of the source activity.

It is the programmer responsibility to assign the security level to data d. Consequently, the level of the transmitted data will be added to the syntax of the method call (see Table 3). Even if it is not detailed in the following, a default behavior should consist in assigning the level of the sender activity to data d sent as request parameter. In turn, every value r returned in a reply transfer will automatically be

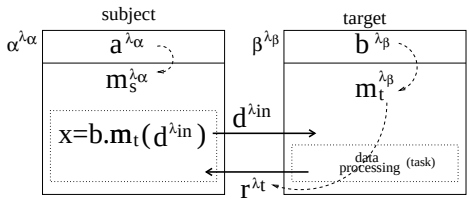

Fig. 2. Communication between security-marked activities.

tagged with the security level of the target method (level inherited from the activity). This form of tagging allows output data to be independent of input data in the processing method, in other words, the security level for the output data does not depend on the level of the input data but on the processing of the data itself.

The conditions for secure communications in ASP are then derived and formalized according to our policy for communications:

Definition 1 (Secure activity creation)

$$\forall \alpha, \gamma \in \mathcal{S}, (\alpha, \gamma, Nw(\gamma, \lambda_\gamma)) \in \mathcal{T} \Longleftrightarrow (\lambda_\alpha \leq \lambda_\gamma) \vee Nw(\gamma, \lambda_\gamma) \in \mathcal{M}(\alpha, \gamma)$$

An activity creation $Nw(\gamma, \lambda_\gamma)$ is authorized if the activity is created with a level greater or equal to the one of the source activity. Else, if α wants to downgrade the data (i.e., the objects) used to create this new activity then there must be an explicit right allowing such an operation.

Definition 2 (Secure request transmission)

$$\forall \alpha, \beta \in \mathcal{S}, (\alpha, \beta, Rq_{\alpha\to\beta}(d, \lambda_{in})) \in \mathcal{T} \Leftrightarrow (\lambda_{in} \leq \lambda_\beta) \wedge \begin{pmatrix} ((\lambda_\alpha > \lambda_{in}) \wedge Rq_{\alpha\to\beta}(d, \lambda_{in}) \in \mathcal{M}(\alpha, \beta)) \\ \vee\ (\lambda_\alpha \leq \lambda_{in}) \\ \vee\ \exists \gamma, \delta, f_i,\ d = fut(f_i^{\gamma\to\delta}) \end{pmatrix}$$

The request transmission $Rq_{\alpha\to\beta}(d, \lambda_{in})$ is authorized if the security level λ_{in} of the transmitted data d, is smaller or equal to the security level λ_β of the target activity β; or, when source activity α with level λ_α tries to assign a level λ_{in} to data d (i.e. a data downgrading), there is an explicit right (discretionary rule $\mathcal{M}(\alpha, \beta)$) that grants it. The philosophy behind a secure request transmission is to "release" information only to a target which holds the appropriate clearance.

We also have a safe request transmission if the security level of data λ_{in} is greater or equal than that of the source λ_α. In that case, we have $\lambda_\alpha \leq \lambda_{in} \leq \lambda_\beta$, showing that activity α safely releases data d because d has a greater security level, and at the same time, activity β receives a lower level data.

Moreover, a safe request transmission is also achieved when handling future references $fut(f_i^{\gamma\to\delta})$ as data. Future references can be freely transmitted between activities because they do not hold any valuable information. We recall that values associated to futures hold information but future references only hold addresses or directions pointing to futures. In this sense, if a future reference is known, it does not mean we can directly get the future value, because anyway, the future value

transmission will be performed by, and submitted to the security rules of, a *secure reply transmission*.

Definition 3 (Secure reply transmission)

$$\forall \alpha, \beta \in \mathcal{S} : (\alpha, \beta, Rp_{\beta\to\alpha}(r)) \in \mathcal{T} \iff (\lambda_\beta \leq \lambda_\alpha) \vee (\exists \gamma, \delta, f_i,\ r = fut(f_i^{\gamma\to\delta}))$$

The secure reply transmission REPLY $Rp_{\beta\to\alpha}(r)$ is authorized if the security level λ_β of target β is smaller or equal to the security level λ_α of subject α, or if the transmitted result r only consists of a reference to a future $f_i^{\gamma\to\delta}$.

Table 3 shows the differences between the secure ASP calculus and the original one (Table 1): security information is added to the activation and method call terms.

$$\begin{array}{rll} a, b \in L' ::= & a.m_j(b^{\lambda_{in}}) & \text{method call,} \\ | & Active^{\lambda_a}(a, m_j) & \text{object activation,} \\ | & \ldots & \end{array}$$

Table 3
Secure ASP calculus: modified primitives

After attaching a security level to each activity, parallel confi gurations are now of the following form: $P, Q ::= \alpha^{\lambda_\alpha}[a; \sigma; \iota; F; R; f] \| \beta^{\lambda_\beta}[\cdots] \| \cdots$

$$\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = copy(\iota'', \sigma) \qquad (\alpha, \gamma, Nw(\gamma, \lambda_\gamma)) \in \mathcal{T}\end{array}}{\begin{array}{c}\alpha^\lambda[\mathcal{R}[Active^{\lambda_a}(\iota'', m_j)]; \sigma; \iota; F; R; f] \parallel P \longrightarrow \\ \alpha^\lambda[\mathcal{R}[\iota']; \sigma'; \iota; F; R; f] \parallel \gamma^{\lambda_a}[\iota''.m_j(); \sigma_\gamma; \iota''; \emptyset; \emptyset; \emptyset] \parallel P\end{array}} \text{(SecNewAct)}$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\to\beta} \text{ new future} \\ \iota_f \notin dom(\sigma_\alpha) \qquad \sigma'_\beta = Copy\&Merge(\sigma_\alpha, \iota' ;\ \sigma_\beta, \iota'') \\ \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\to\beta})\} :: \sigma_\alpha \qquad (\alpha, \beta, Rq_{\alpha\to\beta}(\sigma_\alpha(\iota'), \lambda_{in})) \in \mathcal{T}\end{array}}{\begin{array}{l}\alpha^{\lambda_\alpha}[\mathcal{R}[\iota.m_j(\iota'^{\lambda_{in}})]; \sigma_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P \longrightarrow \\ \alpha^{\lambda_\alpha}[\mathcal{R}[\iota_f]; \sigma'_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma'_\beta; \iota_\beta; F_\beta; R_\beta :: [m_j; \iota''; f_i^{\alpha\to\beta}]; f_\beta] \parallel P\end{array}} \text{(SecRequest)}$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = fut(f_i^{\gamma\to\beta}) \qquad F_\beta(f_i^{\gamma\to\beta}) = \iota_f \\ \sigma'_\alpha = Copy\&Merge(\sigma_\beta, \iota_f ;\ \sigma_\alpha, \iota) \qquad (\beta, \alpha, Rp_{\beta\to\alpha}(\sigma_\beta(\iota_f))) \in \mathcal{T}\end{array}}{\begin{array}{c}\alpha^{\lambda_\alpha}[a_\alpha; \sigma_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P \longrightarrow \\ \alpha^{\lambda_\alpha}[a_\alpha; \sigma'_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P\end{array}} \text{(SecReply)}$$

Table 4
Secure parallel reduction rules

Finally, Table 4 presents the semantics of the secure parallel ASP calculus. These semantics rules ensure secure information fbw. They use the security information attached to the activation and method call terms (λ_a and λ_{in} in the

$Nw(\gamma, \lambda_a)$ and $Rq_{\alpha\to\beta}(d, \lambda_{in})$ rules) to verify the secure transmission and activity creation defined before (Definitions1, 2 and 3). When a communication is not authorized, from the formal point of view, it is simply blocked. In practice a dedicated exception should be raised and appropriately handled.

3.2 Secure Information Flow in the Object Model

We formally define the notion of information flow between activities. The considered entities are activities together with their passive objects, and not passive objects on their own. Because activities can be distributed, Non-Interference related notions [12] can not be directly applied to our model.

Next, system-wide information flows are described by a path. The path is the route along which the information travels, it is constructed by a chain of communicating activities where a subject activity is the starting-point and a target activity is the end-point of the path. Each information transmission observed on each activity will serve for the construction of a path. This path will be called *flow-path*.

Flow-paths fp are lists of activities ($fp := \alpha.\beta.\cdots$). They consist of the ordered list of transiting activities for a given information flow. For example $\varphi_{\gamma.\delta}(\alpha, \beta)$ means that some information has been transmitted from activity α to activity β through activities γ and δ. Concatenation of flow-paths fp and fp' is denoted by $fp.fp'$. By application of the security mechanisms to the non-secure information flow and flow-paths, a first property results: previous definition of information flow for an activity becomes secure if all activity creations, requests and replies transmissions are secure.

Definition 4 (Secure information flow) *An* elementary flow of information *is either based on the sending of a request, or on the sending of a reply, or on the creation of an activity. A* flow of information *is sequentially composed of several elementary flows. The flow-path of any flow of information is the concatenation of intermediate activities, it allows us to retrieve the original elementary flows. Secure information flow is built by concatenation of* elementary secure information flows *which are secured communications: secure* REQUEST, REPLY, *or* NEWACT. *Formally:*

$$\frac{(\alpha, \beta, Rq_{\alpha\to\beta}(\sigma(\iota'), \lambda_{in})) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\alpha, \beta)} \qquad \frac{(\beta, \alpha, Rp_{\beta\to\alpha}(\sigma_{\alpha}(\iota_f))) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\beta, \alpha)}$$

$$\frac{(\alpha, \gamma, Nw(\gamma, \lambda_{\gamma})) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\alpha, \gamma)} \qquad \frac{Sec\varphi_{fp_1}(\alpha, \gamma) \qquad Sec\varphi_{fp_2}(\gamma, \beta)}{Sec\varphi_{fp_1.\gamma.fp_2}(\alpha, \beta)}$$

The following property states that a flow of information is secured if and only if it follows a secure path.

Property (Secure path for information flow) *A flow of information is secured if and only if it is composed of elementary secure information flows.*

$$Sec\varphi_{\gamma_1 \cdots \gamma_n}(\alpha, \beta) \Longleftrightarrow Sec\varphi_{\emptyset}(\alpha, \gamma_1) \wedge Sec\varphi_{\emptyset}(\gamma_1, \gamma_2) \wedge \cdots \wedge Sec\varphi_{\emptyset}(\gamma_n, \beta)$$

The proof of this property is straightforward; it is obtained by induction on the length of the information flow path and by a case analysis on the rules of Definition 4.

This property does not take advantage of the MAC aspect of our model. Indeed, the same property could have been obtained with a purely DAC approach. This property rather shows that our specific and somehow less restrictive definition of information flow does not compromise secured information flow. A secured information flow property using the MAC aspect of our security policy is beyond the scope of this study. More generally, the study of the relation between mandatory and discretionary rules is closely related to the work of Bertino et al. [5].

Compared with other solutions, our secure information flow is the simple composition of a complex elementary flow. This results from the adaptation of the security formalism to a specific service-oriented framework. Complexity of the elementary flow comes from the asymmetric and asynchronous nature of ASP communications. Once such basic secure communications are ensured, the security of information flows is verified in a simple and intuitive manner. The soundness of secure information flow is thus ensured by a precise definition of *information* in the previous section and the fact that secure communications defined in Section 3.1 ensure that every information flow must verify the security policy.

Also note that the way our discretionary rights matrix is built allows to envision a precise tuning concerning the (dynamic) entities involved and the levels that can be assigned to data when downgrading them.

3.3 Specificity of Service-Oriented Computing

Preceding sections showed how classical MAC and DAC security principles can be adapted to ASP and extended, finally ensuring both flexibility and scalability.

Future references are first class objects and can be passed between activities (feature known as *automatic continuations*), thus they have an important consequence concerning the secured flows of information. Indeed, without automatic continuations, a flow of replies would directly follow the opposite path that a flow of requests. In other words, in a classical mandatory ruled system, a request-reply pattern of communications can only occur between entities that have the same security level.

A first contribution of this paper is to authorize discretionary exceptions to these rules concerning level of data transmitted by requests. This allows some request-reply pattern to occur when the request sends non confidential data.

The possibility to transmit future references leads to a model well adapted specific to service-oriented computing. Let us focus on the configuration of Figure 3.

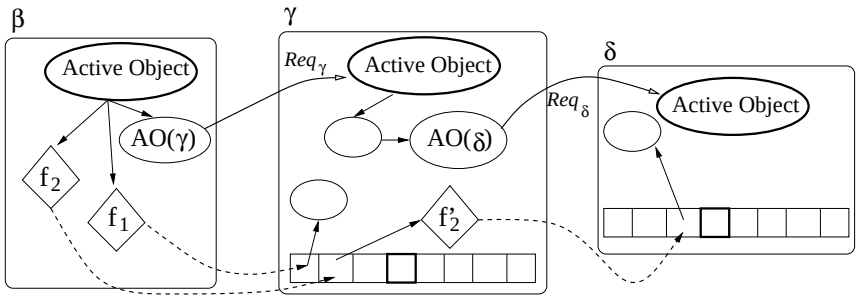

Fig. 3. A classical example: delegation

This configuration can be obtained by a classical scenario: β makes a request to γ (future f_2); but β is only an intermediate and delegates to δ the responsibility to calculate the final result (future f_2'). Let us suppose that $\lambda_\delta \leq \lambda_\beta < \lambda_\gamma$, that is to say the intermediate activity γ has a (too) high security level (this may be the reason why the request has been delegated).

If one did not have automatic continuation, γ could not return the value of future f_2 because it is a future reference to f_2'. Anyway, δ can reply to γ (because $\lambda_\delta < \lambda_\gamma$) but γ cannot forward this result value to β because $\lambda_\beta < \lambda_\gamma$. Indeed, the derivation transmitting the result from δ to β cannot be derived by the preceding rules (**not authorized** means reduction rule cannot be applied):

$$\dfrac{\dfrac{\lambda_\delta < \lambda_\gamma}{(\delta, \gamma, Rp_{\delta\to\gamma}(result)) \in \mathcal{T}} \quad \dfrac{\textbf{not authorized}}{(\gamma, \beta, Rp_{\gamma\to\beta}(r')) \notin \mathcal{T}}}{Sec\varphi_\emptyset(\delta, \gamma) \qquad Sec\varphi_\emptyset(\gamma, \beta)}$$
$$Sec\varphi_\gamma(\delta, \beta)$$

One could expect a better behavior because the direct reply from δ to β should be authorized according to the security model: δ cannot reply only because there is an intermediate activity γ. Indeed if the request had not transited by γ the reply would be authorized. This is why the secured communication rules state that,futures can be freely transmitted (remember a future does not hold valuable information); consequently γ can reply to β if the response is restricted to a future reference. Afterward, δ can reply directly to β because $\lambda_\delta \leq \lambda_\beta$, and β obtains the real value associated to f_2.

$$\dfrac{r = f_2'^{\gamma\to\delta}}{\dfrac{(\gamma, \beta, Rp_{\gamma\to\beta}(r)) \in \mathcal{T}}{Sec\varphi_\emptyset(\gamma, \beta)}} \qquad \dfrac{\lambda_\delta \leq \lambda_\beta}{\dfrac{(\delta, \beta, Rp_{\delta\to\beta}(r)) \in \mathcal{T}}{Sec\varphi_\emptyset(\delta, \beta)}}$$

This example justifies the possibility to freely transmit future references and demonstrates a communication pattern that would not be possible without the expressiveness of futures and the specific secured rules that exist in our mechanism. Whereas, in ASP, the order in which future update occur has no consequence on

the execution of a program, this example shows that in Secure ASP it is important to adopt a convenient future update strategy.

4 Related Work

Henessy and Riely present an extension of π-calculus [18,19], a calculus aimed at distributed systems, extended through the use of (security) types [14]. We do not employ explicit channels to communicate but the read and write actions are analogous to receiving or sending requests and replies (a read when the request or reply is received, and a write when the request or reply is sent). Additionally, their processes may be analogous to our activities, but in general the security policies are not compatible nor they can be encoded in our model even with analogous notions.

Bertino et al. [5] treat exception-based information flow controls in object-oriented systems. They extend close work from Jajodia, Kogan, Sandhu [17] and Samarati et al. [23] to include operations (exceptions) normally not allowed by the strict security policy. They use an ACL (discretionary control) to operate on write and create actions, and with the permissive exceptions, they relax the strict policy imposed on those same actions. The use of exceptions to alter the strict applications mandatory rules of [5] is similar to the use of discretionary conditions in our framework. Both mechanisms allow one to bypass the rigorous application of strict/mandatory access controls.

Attali, Caromel and Contes present high-level rules which define a security policy for GRID applications built upon ProActive [2]. It is based on a discretionary approach where entities follow a hierarchical structure and relies on a Public Key Infrastructure. By comparison, our work focuses only on the communication actions, studies confidentiality in information flows specific to service-oriented applications, and uses both discretionary and mandatory approaches.

On the practical (implementation) side, Java-like languages that include information flow controls (as in [3,20]) could be complemented with our model. They control information flows inside a program, so they could be enhanced to control all communication interactions with other local and non-local programs, in either distributed or cooperative systems.

Comparing to previous work, Definitions 1 and 2 put together a combination of independent mandatory and discretionary controls, but the discretionary condition is evaluated only when the transmission involves a data downgrading case. This does not represent a standard way of policy enforcement because we also have the ability to verify the security level of a new activity (i.e. a whole entity which encapsulates data), and of a single transmitted data (i.e. data sent in requests). Moreover, definition 4 is implicitly based on the Bell-LaPadula model, where security at run-time takes into account a state machine with an initial secure state, and whose transitions from state to state are secure if the access rules are not violated.

5 Concluding Remarks and Future Work

We have presented a precise security model for the secure information flow in ASP. The solution is mainly founded on three cornerstones: the concept of flow of information, security levels attached to activities, and the definition of security rules to be applied to all communications.

The security policy (involving assignation, use, and definition of security levels) also takes into account the way information is handled in our model. When confidentiality is involved, there may exist high-level activities which may need to communicate with low-level activities (normally denied by the mandatory access rules of "no write down"). So by also tagging data with a security level we gain flexibility as the mandatory rules are not broken, and still, with the help of additional discretionary rules, we guarantee that this kind of actions are explicitly allowed. Communications are then controlled according to specific security rules. These rules are predefined in the case of mandatory rules, where the security levels of processes and data are always compared and applied; and in the case of discretionary rules, they are externally defined (for example in a file describing permissions for the whole system). By allowing discretionary exceptions to the mandatory rules, we obtain a highly expressive system. Consequently, the resulting properties are rather weak as we do not take advantage of the MAC aspect of our model; however, stronger properties could be guaranteed by imposing some consistency requirements to the DAC rules.

Finally, we have demonstrated the specificity of our security mechanism: its application to service-oriented computing with replies by the means of futures, and the specific issues that are arisen by our communication mechanism.

A prototype of this security model has been implemented in the ProActive middleware [21] for distributed and mobile (Grid) computing, and is currently under evaluation on real-size examples for scalability and flexibility. In the future, it is planned to study more specifically moblility aspects and to use a role-based access control (RBAC) approach in order to extend and improve the discretionary access to activities.

References

[1] Abadi, M. and L. Cardelli, "A Theory of Objects," Springer-Verlag, 1996.

[2] Attali, I., D. Caromel and A. Contes, *Hierarchical and declarative security for grid applications*, in: *International Conference On High Performance Computing, HIPC, Hyderabad, India, December 17-20* (2003).

[3] Banerjee, A. and D. A. Naumann, *Using access control for secure information flow in a java-like language*, in: *16th IEEE Computer Security Foundations Workshop (CSFW-16)*, 2003.

[4] Bell, D. E. and L. J. LaPadula, *Secure computer system: Unified exposition and*

multics interpretation, Technical Report MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA (1976).

[5] Bertino, E., S. D. C. di Vimercati, E. Ferrari and P. Samarati, *Exception-based information flow control in object-oriented systems*, ACM Transactions on Information and System Security (TISSEC) **1** (1998), pp. 26–65.
URL http://seclab.dti.unimi.it/Papers/tissec98.ps

[6] Caromel, D. and L. Henrio, "A Theory of Distributed Objects," Springer-Verlag, New York, 2005.

[7] Caromel, D., L. Henrio and B. Serpette, *Asynchronous and deterministic objects*, in: *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004* (2004), pp. 123–134.
URL http://www-sop.inria.fr/oasis/Proactive/doc/ASP.ps

[8] Caromel, D., W. Klauser and J. Vayssiere, *Towards seamless computing and metacomputing in java*, Concurrency: Practice and Experience **10** (1998), pp. 1043–1061.

[9] Cortesi, A. and R. Focardi, *Information flow security in mobile ambients*, in: *International Workshop on Concurrency and Coordination (ConCoord'01)*, Electronic Notes on Theoretical Computer Science **54** (2001).
URL
http://www.elsevier.nl/gej-ng/31/29/23/87/27/32/54006.ps

[10] Crafa, S., M. Bugliesi and G. Castagna, *Information flow security in boxed ambients*, , **66:3** (2002).

[11] eXtensible Access Control Markup Language, X., *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml*.
URL
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[12] Focardi, R. and R. Gorrieri, *Classification of security properties (part i: Information flow)*, in: *Foundations of Security Analysis and Design (FOSAD 2000) - Tutorial Lectures*, Lecture Notes in Computer Science **2171** (2001), pp. 331–396.

[13] Hennessy, M., *The security picalculus and non-interference*, Journal of Logic and Algebraic Programming (2003), to Appear.

[14] Hennessy, M. and J. Riely, *Information flow vs. resource access in the asynchronous pi-calculus*, Computer Science Technical Report 2000:03, The University of Sussex (2000).

[15] Herrmann, P., *Information flow analysis of component-structured applications*, in: *17th Annual Computer Security Applications Conference*, The Applied Computer Security Associates (ACSA), 2001.

[16] Honda, K., V. Vasconcelos and N. Yoshida, *Secure information flow as typed process behaviour*, in: *Programming Languages and Systems*, Lecture Notes in Computer Science **1782** (2000).

[17] Jajodia, S., B. Kogan and R. S. Sandhu, *A multilevel secure object-oriented data model*, in: M. D. Abrams, S. Jajodia and H. J. Podell, editors, *Information Security: An Integrated Collection of Essays*, IEEE Computer Society Press, 1995 pp. 596–616.

[18] Milner, R., "Communicating and Mobile Systems: the π-Calculus," 1999.

[19] Milner, R., J. Parrow and D. Walker, *A calculus of mobile processes, part I/II* **100** (1992), pp. 1–77.
URL
http://www.lfcs.informatics.ed.ac.uk/reports/89/ECS-LFCS-89-85/

[20] Myers, A. C., *Jflow: Practical mostly-static information flow control*, in: *26th ACM Symposium on Principles of Programming Languages (POPL 99)* (1999), pp. 228–241.

[21] ProActive, *http://www-sop.inria.fr/oasis/proactive/*.
URL http://www-sop.inria.fr/oasis/ProActive/

[22] Sabelfeld, A., *The impact of synchronisation on secure information flow in concurrent programs*, in: *4th International Conference on Perspectives of System Informatics*, Lecture Notes in Computer Science **2244** (2001).
URL http://www.cs.cornell.edu/~andrei/sabelfeld-psi01.ps

[23] Samarati, P., E. Bertino, A. Ciampichetti and S. Jajodia, *Information flow control in object-oriented systems*, IEEE Transactions on Knowledge and Data Engineering **9** (1997), pp. 524–538.

[24] Samarati, P. and S. D. C. D. Vimercati, *Access control: Policies, models, and mechanisms*, in: *Foundations of Security Analysis and Design : Tutorial Lectures*, Lecture Notes in Computer Science **2171** (2001), p. 137.
URL http://seclab.crema.unimi.it/Papers/sam-fosad.ps

[25] Zdancewic, S., L. Zheng, N. Nystrom and A. C. Myers, *Untrusted hosts and confidentiality: Secure program partitioning*, , **35** (2001), pp. 1–14.

A An Example

A.1 Context

We focus an a financial application specifically oriented to the stock market. The global scenario has the following actors: a stock exchange market, a few banks, and some clients. In this scenario, the main idea is to transform some source information in order to produce another type of information latter used as support in a trade decision process (e.i. decisions to make with respect to the buying and selling of actions in the stock market). In this sense, the stock exchange market will act as the provider of the raw or source information, transferring it to its direct clients, which in this case are the banks.

By unfolding the general bank process (Figure A.1), we notice that it is composed of other subprocesses. Figure A.2 exposes with the BPMN notation the

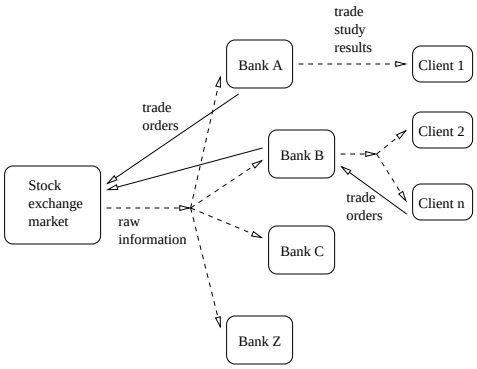

Fig. A.1. Generalized business schema for a financial application.

composition of a bank process. A reception process takes the information coming from the stock exchange, and according to some filtering and redistribution rules, it sends the possible narrowed information to its specified targets. The targets may be external business partners or they can be internal.

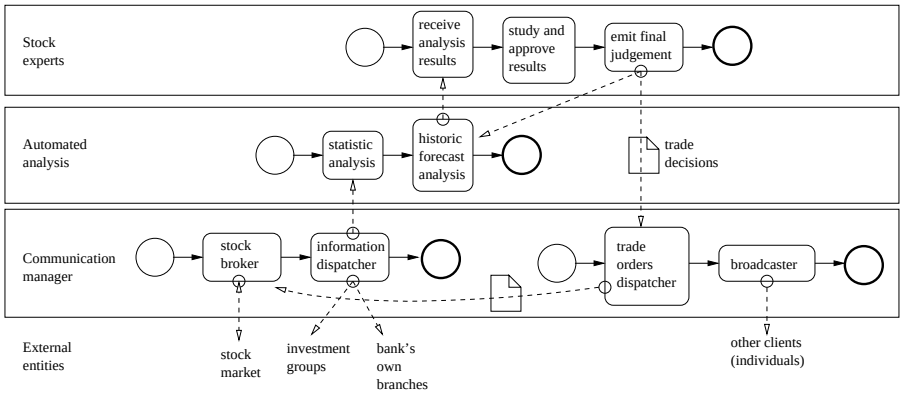

Fig. A.2. A bank's private business processes.

Following the bank's internal or private workflow, after reception of the information by the stockbroker process, the provided information is passed to a statistic analysis, then confronted with historic forecasts, the result studied by market experts, producing buying and selling orders, which are finally sent to the stock exchange through the same initial stockbroker process. Finally, there is a new redistribution of information executed by the trade orders dispatcher. In this new information distribution, the whole or part of the analysis results can be offered and sold to other kind of clients, meanwhile the stockbroker places the trade orders to the stock market. Moreover, the stockbroker process has two important activities, one is the reception of information, and the other is the reply to this reception with the trade orders.

A.2 An Implementation in Secured ProActive

The internal processes presented in the previous section are grouped into general processes corresponding to ProActive *activities* named *E* (experts), *A* (analysis), *C1*

and *C2* (Communications), *S* (stock), *I* (investment), *B* (branch), and *Clnt* (client).

Figure A.3 shows the information flows translated into a ProActive communication schema (based on REQUESTS –*Rq*– and REPLIES –*Rp*).

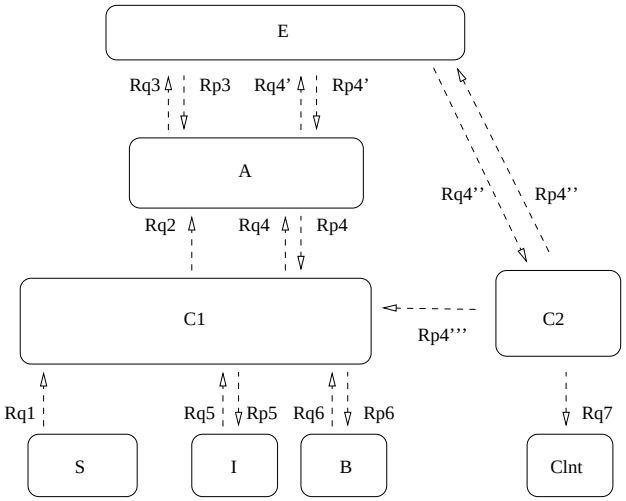

Fig. A.3. ProActive model for the banking process, with request and reply transmissions.

In the case of *Rq1*, *S* provides information to *C1* in a "push" manner (through a repetitive transmissions of requests), a constant feeding of information can be done, and in consequence, there is no need of replies from *C1* to *S*.

Emission of information from *C1* to *A* is achieved with *Rq2*. Exchange of information between *A* and *E* is done with the pair *Rq3-Rp3*.

The communication between *C2* and *C1* requires a series of transmissions. *C1* will require the results from *C2* and will have to communicate through *A* and *E*. Then, with *Rq4*, *C1* asks the results to *A*; with *Rq4'*, *A* demands the results to *E*; in turn, with *Rq4"*, *E* communicates with *C2*; and finally, the response will return to *C1* trough the corresponding replies.

Information from *C1* to *I* and *B* is provided on demand, hence the respective pairs *Rq5-Rp5* and *Rq6-Rp6*. *Rq7* between *C2* and *Clnt* follows the same methodology as *Rq1*. Finally, transactions of trade orders between *C1* and *S* are accomplished with *Rq8-Rp8*.

From Figure A.3, the assignation of security levels is to be done, and depending on these affectations, entries for matrix $\mathcal{M}$ have to be added.

For the bank's internal processes, let us define a relation of security levels with $\lambda_E = \lambda_A > \lambda_{C1} > \lambda_{C2}$ meaning that, process *E* gets the highest security level because it represents the financial expert's work; process *A* also gets a high level because it saves the results coming from *E* for the historic analysis; and process *C1* gets the next highest level after *A* mostly because it has to manipulate the results or trade decisions. With respect to the security levels for the bank's external processes, we suppose: $\lambda_S > \lambda_{C1}$ (the stock market requires to have a high level of protection), $\lambda_I > \lambda_{C1}$ (the investment group requires also a high level), $\lambda_B = \lambda_{C1}$

(considering the bank's branches are "internal" processes), and $\lambda_{C2} > \lambda_{Clnt}$ (the bank has a higher level of security than the clients).

Once the security levels are assigned to processes, REQUESTS are analyzed in order to determine what are the levels to be assigned to data transmitted, corresponding entries are also required for the discretionary access.

Request	Level of data	Entry for $\mathcal{M}(\alpha, \beta)$	Reason for reply
Rq_1	$\lambda_{in} = \lambda_{C1}$	$Rq_{S \to C1}(d, \lambda_{C1})$	—
Rq_2	$\lambda_{in} = \lambda_{C1}$	—	—
Rq_3	$\lambda_{in} = \lambda_A$	—	$\lambda_E = \lambda_A$
Rq_4	$\lambda_{in} = \lambda_{C1}$	—	$r = fut(f_4''^{E \to C2})$ or $r = fut(f_4'^{A \to E})$
			Rp_4''' authorized by $\lambda_{C2} < \lambda_{C1}$
$Rq_{4'}$	$\lambda_{in} = \lambda_A$	—	$r = fut(f_4''^{E \to C2})$
$Rq_{4''}$	$\lambda_{in} = \lambda_{C2}$	$Rq_{E \to C2}(d, \lambda_{C2})$	$\lambda_{C2} < \lambda_E$
Rq_5	—	—	$\lambda_{C1} = \lambda_I$
Rq_6	—	—	$\lambda_{C1} = \lambda_B$
Rq_7	$\lambda_{in} = \lambda_{Clnt}$	$Rq_{C2 \to Clnt}(d, \lambda_{Clnt})$	—

Table A.1
Summary of security levels assigned to the bank's processes

Table A.1 gives the complete overview of security levels given to data, the required entries for matrix $\mathcal{M}$, and the values returned in the responses. From this table, in the case of request Rq_1, by assigning to the transmitted data the same security level of the target (λ_{C1}), it is needed an entry or explicit rule, allowing communications from S to $C1$.

For request Rq_2 there is no needed entry for the matrix, and also there is no reply.

In the case of request Rq_3, there is no needed entry, and the corresponding reply will be the value of r (i.e. whatever value process E will return as result for this request). It is also shown the condition that allows the reply, which for Rq_3 is because $\lambda_E = \lambda_A$.

For the combined case of the "fourth" request, replies to Rq_4, $Rq_{4'}$, and $Rq_{4''}$ will contain the reference to a future (corresponding to the value to be produced by $C2$). Consequently the value of reply to $C1$ must be obtained directly from $C2$ by another reply $Rp4'''$, and hence allowed to be transmitted because $\lambda_{C2} < \lambda_{C1}$. Other replies must only contain informations stating aliasing between futures (e.g. the value of future $f_4'^{A \to E}$ is the future $f_4''^{E \to C2}$). Note that aother possible replies go from E to $C1$, from $C2$ to A, but a reply containing the result from A to $C1$ is forbidden, thus $C1$ must first receive aliasing informations stating that the future $f_4^{C1 \to A}$ is the future $f_4''^{E \to C2}$ before receiving $Rp4'''$.

For requests Rq_5, and Rq_6, there is no data transmitted in the request but they

both receive a reply value.

Rq_7 is similar to Rq_1.

This example shows the specifi city of our security model. Considering secure path for information fbw from $C1$ to $C2$, formally written $Sec\varphi_{A.E}(C1, C2)$, the fbw of information transits through high-level processes (i.e. processes A and E). Classically, these high-level process would block any subsequent communication because of the "no write down" property (e.g. request $Rq_{4''}$ is sending (or writing) data from a high to a low level process, and in consequence, it would be blocked). Nevertheless, the security model allows a safe communication from end-to-end (in this case between $C1$ and $C2$).